Fast Continuous Collision Detection and Handling for Desktop Virtual Prototyping

Stephane Redon

Fast Continuous Collision Detection and Handling for Desktop Virtual Prototyping

Stephane Redon

i3D - INRIA - France* †

Abstract

This paper presents an overview of our recent work on continuous collision detection methods and constraints handling for rigid polyhedral objects. We demonstrate that continuous collision detection algorithms are practical in interactive dynamics simulation of complex polyhedral rigid bodies and show how continuous collision detection and efficient constraint-based dynamics algorithms allow to perform various virtual prototyping tasks *intuitively, precisely and robustly* on commodity desktop computers. Especially, we present two applications of our system to actual industrial cases. We note that both tasks are performed with a simple 2d mouse on a high-end computer.

1 Introduction

Computer aided design and manufacturing has been a very active research field for many years, which has contributed to the development of major product lifecycle man-

*The author is currently affiliated with the University of North Carolina at Chapel Hill.

†**Correspondence and offprint requests to:** Stephane Redon, Department of Computer Science, 336 Sitterson Hall, CB ♯ 3175 University of North Carolina Chapel Hill, NC 27599-3175. Email: redon@cs.unc.edu.

agement softwares.

One important aspect of product lifecycle management is *virtual prototyping*, which aims at designing manufactured products through *digital mock-ups*, in order to suppress the need for expensive real mock-ups. Because the analysis and simulation of a product through a digital mock-up must provide insight on the product *before* it is manufactured, virtual prototyping requires to be able to simulate the physical properties of the objects being designed.

This paper focuses on a fundamental requirement of virtual prototyping: the ability to *detect* and *handle* collisions between virtual objects. Typical applications of collision detection and handling in virtual prototyping include path-planning, tolerance verification and assembly/disassembly tasks conception.

Most well-known collision detection methods are *discrete*: they sample the objects' trajectories at discrete times and report *interpenetrations* only (recent surveys include [11, 13]). As a result, they can miss collisions when the objects move rapidly or are small. Moreover, even when an interpenetration has been detected, it is often difficult to reposition the objects in a *contacting* position (*i.e.* with no interpenetration). One reason being that computing the penetration depth for general objects is a difficult problem, which does not take the objects' motion into account. Relying on the penetration depth to reposition the objects might thus lead to the well-known *pop-through* effect, where an object enters an obstacle on one side and goes out of it on the other side, as can be seen in Figure 1.

Even *backtracking methods*, which attempt to compute the first time of collision by recursively subdividing the time interval after an interpenetration has been detected, can fail when the object is not connected. Assume that the current time interval is $[t_n, t_{n+1}]$, and assume an interpenetration has been detected at time t_{n+1} Essentially, one time of first contact t_e is *estimated* in this interval (for example, by taking the midpoint of the

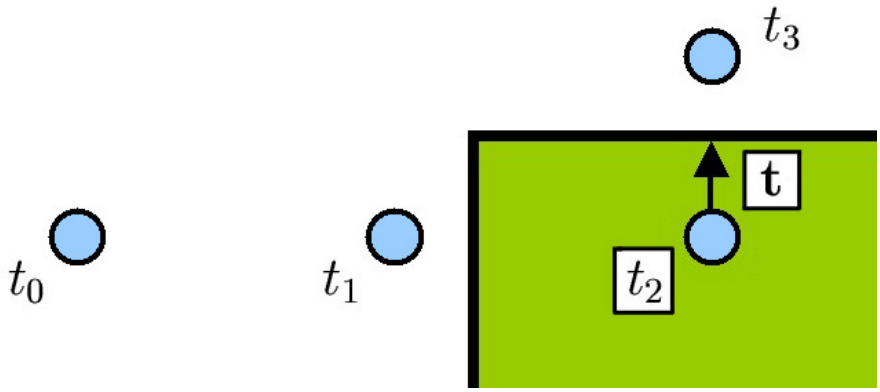

Figure 1: *The pop-through effect*: allowing interpenetration between objects can lead to incoherences in the simulation. In this example, the object penetrates the obstacle at time t_2 and pops out from the wrong side of the obstacle, because of the smaller penetration in the exit direction.

time interval). Objects' positions are then computed at this instant and an interpenetration detection is performed again. Depending on whether the objects interpenetrate or not, the algorithm decides that the first time of collision is in $[t_n, t_e]$ or $[t_e, t_{n+1}]$, respectively, and loops on this new interval. The process stops when the amount of interpenetration is smaller than a predetermined threshold.

The computational cost of backtracking can be high when objects are complex or when they have interpenetrated much. Besides, since backtracking is only performed when an interpenetration has been detected, non-connected objects, or even non-convex objects, can enter a configuration from which they could not get out (e.g. two torii that would become interlocked).

Consequently, discrete collision detection methods cannot guarantee the existence of *complete* interpenetration-free paths, and can lead to inconsistent, unrealistic, or even unreachable states in the virtual scene. This may be a problem for simulation and analysis of digital mock-ups when *provable* information is required. Especially, some *ad hoc* strate-

gies have sometimes to be defined when using a discrete collision detection method in a virtual prototyping context in order to attempt to guarantee interpenetration-free paths. One of them consists in not moving the object when the next position involves a penetration. While this may be sufficient when few obstacles only surround the moving object, this may lead to an impossibility of manipulating the object when the environment is cluttered. Moreover, this method guarantees interpenetration-free *positions* only, as collisions can still be missed. Other strategies consist in using force fields and haptic interface to guarantee that the manipulated object remains far from the environment.

In the following, we present an overview of our recent work on *continuous* collision detection methods and constraints handling for rigid polyhedral objects. We demonstrate that continuous collision detection algorithms are possible in interactive dynamics simulation of rigid bodies and show how continuous collision detection and constraint-based dynamics algorithms allow to perform various virtual prototyping tasks *intuitively, precisely and robustly* on commodity desktop computers. Especially, we present two applications of our system to actual industrial cases. We note that both tasks are performed with a simple 2d mouse.

2 Continuous collision detection

2.1 Arbitrary in-between motion

Despite their shortcomings, it is understandable why discrete collision methods have been developed and used to such a large extent in dynamics simulations. Besides their assumed lower cost, it must be noticed that, most of the time, *the actual objects' motions are not known*. One important reason is that the differential equations governing the system's dynamics are solved using discretized methods (*e.g.* Euler or Runge-Kutta schemes). Another reason is that, in an interactive application, the user interface (*e.g.* the

mouse, or a haptic device) sends the user's actions to the system at discrete instants only. Other interfaces such as tracking systems, which are not necessarily interactive, also provide discretized data only. For all these reasons, it is most of the time impossible to have a closed-form expression of the objects' motions, and the objects' positions, velocities and accelerations are available at discrete times only.

Because the actual motion of an object is not known, we have proposed to replace it by an *arbitrary in-between motion*, which interpolates successive available data to produce a continuous object motion [20, 23]. Thanks to the interpolation, the global aspect of the objects' trajectories is conserved, while the *local* motion (the unknown one, between two available positions) is modified. The continuous object motion is used both for collision detection and object positioning when a collision has been detected, to ensure that the scene is always in a consistent state, without any interpenetration. The choice of the in-between motion depends on the application. It must be general enough to satisfy the various constraints imposed by the application (*e.g.* objects rigidity, ability to reach any configuration required in the application), but at the same time must be such that the continuous collision detection equations can be solved very efficiently. Any motion meeting these requirements can be *arbitrarily* chosen. Note that the interpolation order depends on the application as well. While it may be sufficient to interpolate positions only, some applications may require higher-order interpolations.

It can be argued that using an interpolating motion which has been arbitrarily chosen does not allow to determine the exact collision times and positions corresponding to the actual objects' motions. Obviously, using an arbitrary in-between motion only allows to report those collisions which occur *when the objects move according to this arbitrary motion*. However, by using the same arbitrary motion to place the objects at the time of collision, the virtual scene is always in a consistent state, and *no interpenetration ever occurs*. Moreover, this method guarantees interpenetration-free *paths*, and not only con-

sistent positions at discrete times.

2.2 Elementary tests

Since we perform continuous collision detection tests, only three types of contact can occur between two general polyhedral objects i and j:

- An edge of i contacts an edge of j
- A face of i contacts a vertex of j
- A vertex of i contacts a face of j

The last two contact types are equivalent and we thus have to perform two different types of *elementary tests*: edge/edge or vertex/face tests. The continuous collision detection equations for both tests are derived simply. Whatever the motions of objects i and j, a collision occurs between an edge $\mathbf{a}(t)\mathbf{b}(t)$ of i and an edge $\mathbf{c}(t)\mathbf{d}(t)$ of j if there exists a time t for which:

$$\mathbf{a}(t)\mathbf{c}(t).(\mathbf{a}(t)\mathbf{b}(t) \wedge \mathbf{c}(t)\mathbf{d}(t)) = 0 \tag{1}$$

which means that the lines containing the edges are colliding, and if at that time the *edges* (not only the lines) are intersecting. Similarly, a collision occurs between a vertex $\mathbf{a}(t)$ of i (resp j.) and a triangle $\mathbf{b}(t)\mathbf{c}(t)\mathbf{d}(t)$ of j (resp. i) if there is a time t for which:

$$\mathbf{a}(t)\mathbf{b}(t).(\mathbf{b}(t)\mathbf{c}(t) \wedge \mathbf{b}(t)\mathbf{d}(t) = 0 \tag{2}$$

which means that the vertex is inside the plane containing the triangle, and if at that time the vertex is *inside the triangle*. Equations (1) and (2) are the ones which have to be solved efficiently when a specific interpolating motion is chosen. Our initial work has focused on these equations to show that using a specific arbitrary in-between motion derived from a screw motion allowed to reduce them to third-degree polynomial equations when one of

the objects is static [20, 21]. A method proposed by Canny in [7] obtains a similar result with a different motion, but its use of quaternions and of a translation along a different axis than the one of rotation makes it more computationally expensive to determine the coefficients of the polynomials.

2.3 Bounding-volumes hierarchies

In order to avoid performing all possible edge/edge or vertex/face face tests for any object pair, many collision detection algorithms rely on *bounding-volume hierarchies*. Basically, if two objects are enclosed in bounding volumes which do not overlap, then it is known for sure that they do not collide. Hierarchies of bounding volumes are used to recursively perform such overlap tests which can conservatively cull away large parts of the objects when testing for a collision. Typical bounding volumes used for collision detection include spheres [19, 10, 25, 6], axis-aligned bounding boxes (AABBs) [28], oriented bounding boxes (OBBs) [8], and k-dops [9].

For *continuous* collision detection, it is necessary to perform continuous overlap tests between bounding volumes. We have proposed in [23] to extend the discrete OBB/OBB overlap test proposed by Gottschalk *et al.* [8] to the continuous domain by using *interval arithmetic* [17, 26]. The static test relies upon the separating axis theorem. Let us assume that the first OBB is described by three axes $\mathbf{e}_1$, $\mathbf{e}_2$ and $\mathbf{e}_3$, a center $\mathbf{T}_A$, and its half-sizes along its axes a_1, a_2 and a_3. In the same way, the second OBB is described by its axes $\mathbf{f}_1$, $\mathbf{f}_2$ and $\mathbf{f}_3$, its center $\mathbf{T}_B$, and its half-sizes along its axes b_1, b_2 and b_3. The separating axis theorem states that two static OBBs overlap if and only if all of fifteen separating axis tests fail. A separating test is simple: the axis $\mathbf{a}$ separates the OBBs if and only if:

$$|\mathbf{a} \cdot \mathbf{T}_A\mathbf{T}_B| > \sum_{i=1}^{3} a_i|\mathbf{a} \cdot \mathbf{e}_i| + \sum_{i=1}^{3} b_i|\mathbf{a} \cdot \mathbf{f}_i| \tag{3}$$

The sufficient set of fifteen axes is:

$$\{\mathbf{e}_i,\ \mathbf{f}_j,\ \mathbf{e}_i \wedge \mathbf{f}_j,\ 1 \leqslant i \leqslant 3,\ 1 \leqslant j \leqslant 3\} \tag{4}$$

Since each member of the inequality (3) is a function of time depending on the specific arbitrary in-between motion chosen for the application, interval arithmetic can be used to bound both members very efficiently over a time interval $[t_n, t_{n+1}]$. When the lower bound on the left member is larger than the upper bound on the right member, the axis $\mathbf{a}$ separates the boxes during the whole time interval $[t_n, t_{n+1}]$, and the pair of boxes can thus be discarded. Note that basing the continuous overlap test upon bounds computed over a whole time interval produces conservative results only: two moving boxes can be separated over $[t_n, t_{n+1}]$ even when there does not exist *one* axis which separates them on the whole time interval. Because of this, we have proposed a heuristic based upon the boxes' velocities to determine when it may be useful to subdivide the time interval. This allows to cull away more pairs of boxes by a better detection of these situations where more than one axis is required to separate the boxes over the whole time interval.

3 Handling constraints

Closely related to the collision detection problem is the one of handling the geometrical constraints imposed by the collisions which have been detected. In a rigid body dynamics simulation, two different problems have to be solved:

Collision response problem Whenever a new collision occurs, the simulator must determine the objects' *post-impact velocities* from their pre-impact velocities and their dynamic properties.

Constrained motion problem When the collision response problem has been solved, the simulator has to compute the objects' *constrained accelerations* from their un-

constrained accelerations (the ones the objects would have if there weren't any constraints acting on them) and the geometrical constraints imposed by the transient contacts (those for which the relative velocity is zero, *i.e.* those which have a non-zero duration).

Numerous approaches have been suggested to solve both problems and traditional algorithms include penalty methods [18, 12], impulse-based methods [16] and constraint-based methods [14, 3, 4, 27, 1, 2, 15]. Penalty methods are generally used when no precise contact information is available, as they compute the contact forces from the amount of interpenetration between contacting objects. Continuous collision detection naturally provides all the necessary contact information: the contact position, the contacting elements and the contact normal. Consequently, impulse-based or constraint-based methods are a natural choice. We have opted for constraint-based methods because they can handle simultaneous constraints.

Most constraint-based methods formulate both dynamics problems as a linear complementarity problem (LCP) in the *contact-space*, which relates contact forces and accelerations. For example, in the frictionless constrained motion problem, the LCP expresses the relation between the normal contact forces and the relative normal accelerations at the contact points. If $\mathbf{f}$ and $\mathbf{a}_{cp}$ are two vectors in $\mathbb{R}^m$ describing the normal contact forces (in a frictionless system, the contact forces are normal to the contact plane) and the normal relative accelerations to the m contact points, then there exists an $m \times m$ matrix $\mathbf{A}$ and a vector $\mathbf{b}$ in $\mathbb{R}^m$ such as:

$$\begin{cases} \mathbf{a}_{cp} = \mathbf{A}\mathbf{f} + \mathbf{b} \\ \mathbf{a}_{cp} \geqslant 0 \\ \mathbf{f} \geqslant 0 \\ \mathbf{a}_{cp}^T \mathbf{f} = 0 \end{cases} \tag{5}$$

The complementarity relation $\mathbf{a}_{cp}^T \mathbf{f} = 0$ expresses the fact that, for each contact point, either the relative normal acceleration is non-zero (the contact breaks) and then the normal contact force is zero, or the contact force is non-zero (objects remain in contact) and therefore the relative normal acceleration is zero.

The matrix $\mathbf{A}$ is traditional computed from the constraints created by the contact points [3, 24]. Let i and j denote two contacting objects. I is a contact point, $\mathbf{n}$ is the contact normal directed from j to i. Depending on the object it belongs to, I is denoted I_i or I_j.

Note that this distinction is necessary to establish the constraint equations. Although these two points are identical in theory, it is not the case in practice, for example because of the finite precision of the computations. Moreover, we will see in the next section that the method we use to combine the continuous collision detection algorithms to the dynamics algorithms requires us to maintain a distinction between I_i et I_j which become, in practice, the closest points belonging to the polyhedral primitives. In a vertex/face case, for example, I_i is the vertex and I_j is the point in the face which is the closest to I_i.

With this notation, the *non-penetration constraint* on the relative normal acceleration of I is [3]:

$$(\mathbf{a}_i(I_i) - \mathbf{a}_j(I_j)).\mathbf{n} + 2.(\mathbf{v}_i(I_i) - \mathbf{v}_j(I_j)).\frac{d\mathbf{n}}{dt} \geqslant 0 \tag{6}$$

Similarly, a *collision response constraint* on the relative normal velocity can be derived when a collision occurs, to solve the collision response problem:

$$(\mathbf{v}_i^+(I_i) - \mathbf{v}_j^+(I_j)).\mathbf{n} \geqslant -e(\mathbf{v}_i^-(I_i) - \mathbf{v}_j^-(I_j)).\mathbf{n} \tag{7}$$

where e is the restitution coefficient at the contact point.

We have shown in [22] that for frictionless systems Gauss' least constraint principle provides a *motion-space* formulation of both dynamics problems, through a projection

problem which relates the object's accelerations or velocities and the contact forces. Although the two formulations are mathematically equivalent, the motion-space formulation presents several algorithmic benefits: it is better conditioned, always sparse, requires less memory and allows to avoid some redundant computations performed by an algorithm operating in the contact-space. An experimental comparison has suggested that an algorithm operating in the motion-space takes advantage of sparsity to perform increasingly better than a contact-space algorithm as the average number of contact points per object increases. As a result, our system uses a motion-space algorithm based upon Wilhelmsen's projection algorithm [29].

4 Combining continuous collision detection and response

Because we use an arbitrary in-between motion to interpolate successive objects' positions, we need to permanently maintain a small *security distance* between contacting objects. Most of the time, indeed, the arbitrarily chosen in-between motion does not satisfy the dynamics constraints during the successive time intervals.

Consider for example the case of a rectangular box in transient contact with a plane surface, as visible in Figure 2.a. In this example, the contact point I should be permanently on the contact plane $\mathcal{P}$ during the considered time interval $[t_n, t_{n+1}]$. However, the real object motion during this time interval is replaced by an arbitrarily fixed motion which, although it preserves the object's positions at times t_n and t_{n+1}, does not guarantee that the point I remains on the plane $\mathcal{P}$ during the time interval. Depending on the initial and final positions of the box, which impose the in-between motion used for the collision detection, this point I might indeed violate the non-penetration constraint *immediately after* t_n, leading to the detection of a collision at time t_n. In order to avoid that the simulator remains blocked at time t_n or, generally, that it detects these collisions artificially created by the use of an arbitrary in-between motion too often, we introduce a security distance

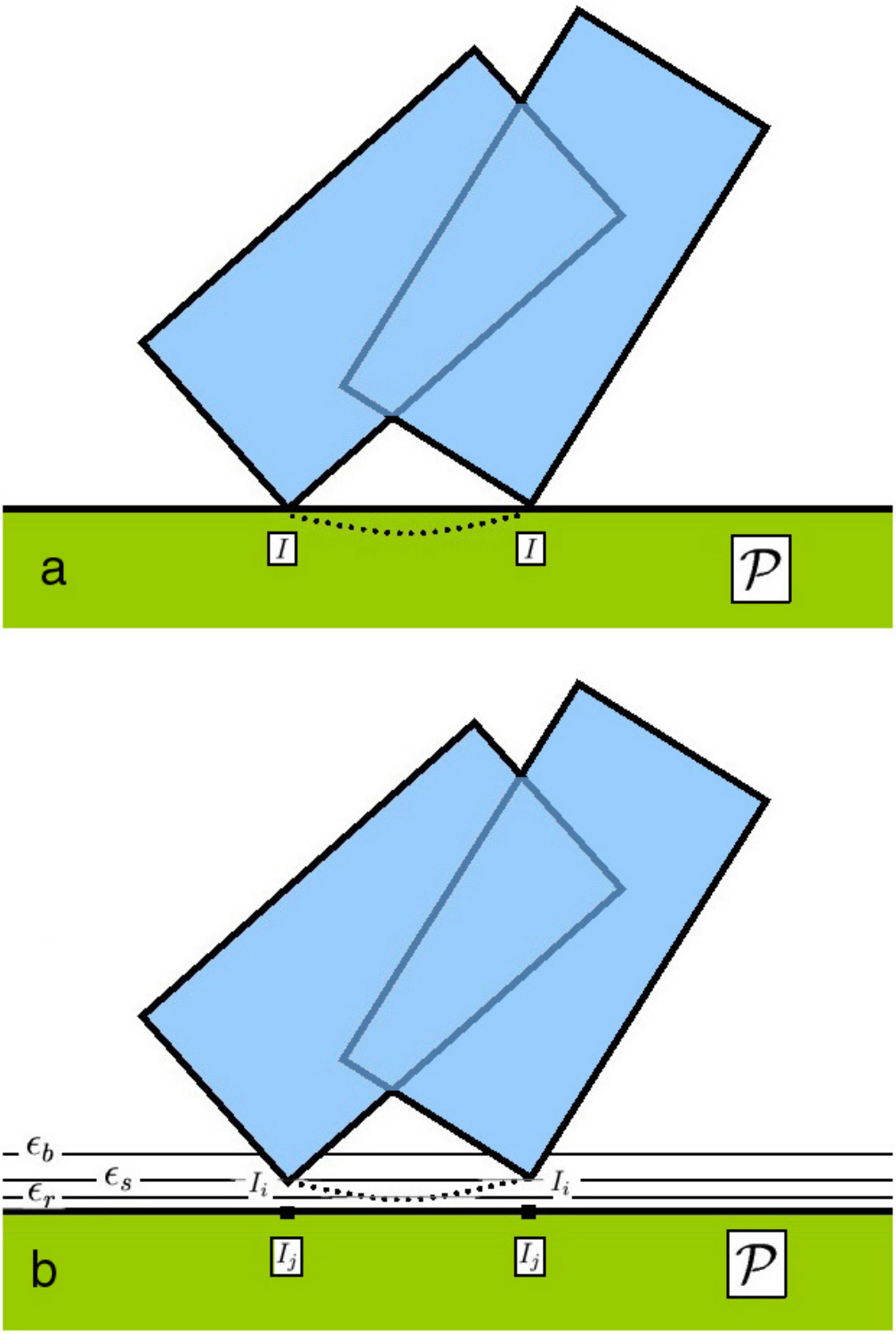

Figure 2: *a*: the arbitrary in-between motion used for the collision detection does not satisfy the non-penetration constraint during the time interval $[t_n, t_{n+1}]$. *b*: contacting objects are maintained slightly distant from each other.

ϵ_s between contacting objects, as shown in Figure 2.b. As a result, we consider that each contact point which has been determined by the continuous collision detection module is characterized by two points I_i and I_j which are the closest points on the polyhedral primitives. These closest points are easily updated whenever the geometrical constraints are modified, as long as a contact is considered to be active.

In order to maintain the security distance between contacting objects over time, the constraints imposed to accelerations or velocities are modified slightly. For example, *when the local distance between the two objects (*i.e. *the one separating the two points I_i and I_j which characterize the contact point) is smaller than the security distance ϵ_s*, the non-penetration constraint (6) becomes:

$$\mathbf{a}_i(I_i) - \mathbf{a}_j(I_j)).\mathbf{n} + 2.(\mathbf{v}_i(I_i) - \mathbf{v}_j(I_j)).\frac{d\mathbf{n}}{dt} \geqslant K + k(\epsilon_s - d) \tag{8}$$

where k is a coupling constant, and d is the distance between the points I_i and I_j.

Two other values, ϵ_b and ϵ_r, are used in combination with the security distance ϵ_s. The first one, ϵ_b, larger than ϵ_s, is the value beyond which the simulator declares that the contact between the two objects is broken (thus removing the need to update I_i and I_j). The second, ϵ_r, smaller than ϵ_s, is an alert value below which the simulator stops the simulation time and enters a *repositioning cycle*, because it did not manage to maintain the objects at a sufficient distance from each other during the simulation. This repositioning cycle, similar to the one introduced by Baraff in [4], is performed between two successive frames by computing the smallest objects' displacements which satisfy the *repositioning constraints*. The repositioning constraints are similar to collision response constraints (7) which become:

$$(\mathbf{v}_i(I_i) - \mathbf{v}_j(I_j)).\mathbf{n} \geqslant \epsilon_s - d. \tag{9}$$

Once these velocities are computed, they are used to move objects (while detecting colli-

sions). As long as there exists a contact for which the local distance is below the repositioning distance ϵ_r, a new repositioning step is done. We have observed that repositioning cycles are generally very short, nearly imperceptible by the user, and do not hinder the progression of the interactive simulation.

Let us note that the repositioning problem is very similar to the collision response problem, since it consists in computing the smallest possible velocities, in the velocity-space, among those which satisfy the repositioning constraints.

In our implementation, objects are scaled so that the total width of the environment is 50 units, and the coupling parameters are $\epsilon_r = 0.0044$, $\epsilon_s = 0.01$, and $\epsilon_b = 0.023$. As a consequence, the security distance and the repositioning effects are not perceived by the user. Let us note that, for environments including mobile objects of very disparate sizes, it is possible to adapt these coupling values to the manipulated object during the interaction, in order to avoid that the size of the object manipulated be of the same order than the security distance. For the databases we tested, however, this was not necessary.

Let us note finally that repositioning objects does not correspond to a physical phenomenon, and can add energy in the system. In order to avoid this, it may be useful to slightly decrease the objects' velocities after a repositioning step. Note however that this problem does not occur in a first-order (quasi-static) simulation, since objects' velocities are zeroed at each frame.

More generally, the repositioning problem is similar to a constraint stabilization problem, for which a classic solution is the one proposed by Baumgarte [5]. In this case, however, constraints are unilateral ones. Moreover, the more difficult problem of unilateral constraint stabilization is greatly facilitated by the continuous collision detection.

5 Results

The algorithms reported here have been integrated to form a portable C++ library, CONTACT Toolkit, which has been successfully tested on Windows and Unix environments. The resulting simulator is able to perform interactive simulations on polyhedral objects which contain up to a few hundreds of thousands of triangles on a 2GHz pentium PC with 256 Megabytes of memory and a NVIDIA GeForce4 MX graphics card.

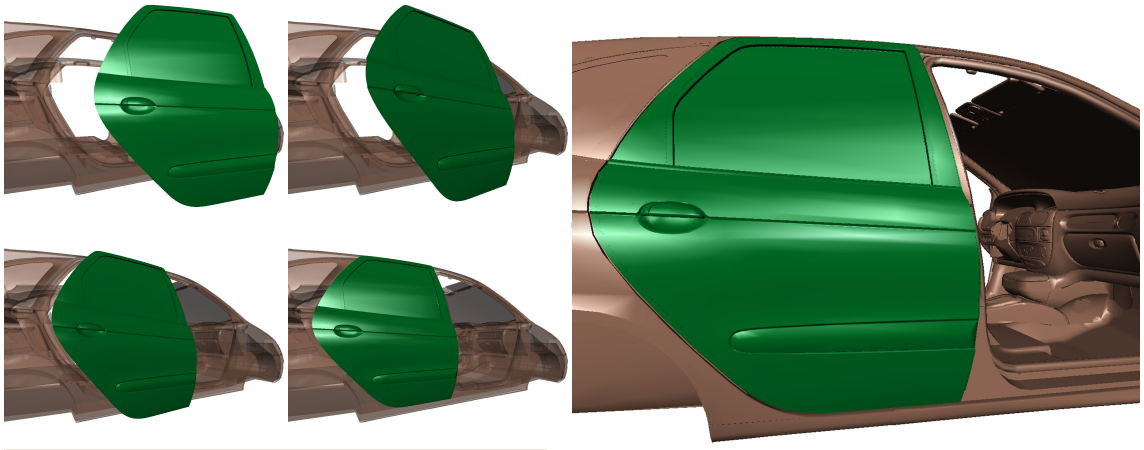

Figure 3: *Interactive positioning of a car door (11,000 triangles) on a car skeleton (130,000 triangles).* Continuous collision detection algorithms and constraint-based dynamics algorithms allow to place the car door precisely and interactively, without any interpenetration ever. 3d models ©Renault.

Here we present two applications of our system to actual industrial databases provided by Renault. We note that for many virtual prototyping tasks it is preferable to perform first-order (quasi-static) simulations, where the objects have no acceleration, as the absence of inertia generally facilitates the manipulation of the objects. This is done by zeroing the objects velocities at each time step and using collision response constraints only [22].

The first application involves positioning a car door (11,000 triangles) into the corresponding car skeleton (130,000 triangles). Figure 3 shows a typical positioning sequence. Our tests demonstrate that a user can easily position the door with a few mouse motions.

The interaction is greatly facilitated by the constraint-based algorithms. The continuous collision algorithms allow the user to precisely position the door, as contacts are detected with the exact objects' geometry, without ever letting them interpenetrate.

The second application consists in removing a window motor from a car door (about 20,000 triangles in the scene). Figure 4 shows a typical interaction sequence. The contact points occurring during the interaction are materialized by small yellow spheres. Again, this application demonstrates the quality of the interaction provided by continuous collision detection algorithms and constraint-based dynamics algorithms. The window motor is very easily removed from the car door with a few mouse motions.

We note that in both cases, the interactive simulator allows to *guarantee* complete interpenetration-free paths, and that both tasks are performed with a simple mouse on a high-end PC.

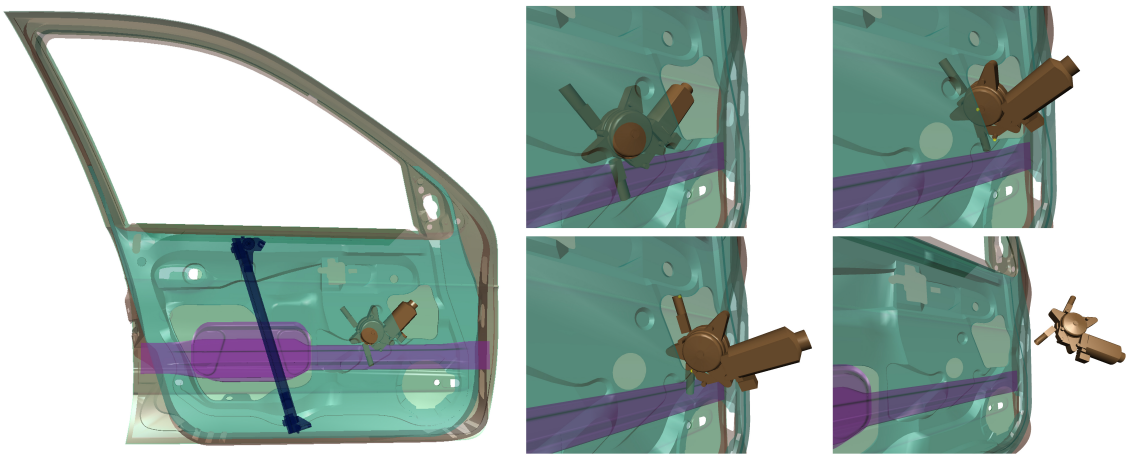

Figure 4: *Determining the existence of a path to remove the window motor from the car door*. The continuous collision detection algorithms allow to guarantee the existence of an interpenetration-free path to remove the motor while interactively and intuitively manipulating the motor thanks to efficient constraint-based dynamics algorithms. 3d models ©Renault.

Our tests also show that a user does not notice the fact that an arbitrary in-between motion is employed to replace the actual objects' motions between successive frames.

6 Conclusion and future work

This paper has presented an overview of our recent work on continuous collision detection and constraint handling methods for polyhedral rigid bodies. We have discussed various shortcomings of discrete collision detection methods, especially in the context of virtual prototyping, and have described a continuous collision detection method based upon *arbitrary in-between motions*. We have shown how this method can be coupled to constraint-based dynamics algorithms. Our tests show that the simulator based upon the algorithms reported in this paper can be used for interactive dynamics simulations of complex polyhedral rigid bodies. Moreover, we have presented the applicability of our algorithms to actual industrial cases provided by Renault, which seem to demonstrate that continuous collision detection algorithms combined to efficient constraint-based dynamics algorithms can be a powerful tool for virtual prototyping tasks.

Future work include in-depth tests of our simulator. The library CONTACT Toolkit is currently being tested at Renault, PSA Peugeot Citroën and Airbus-EADS. This should help to evaluate more precisely the validity and the scalability of the algorithms reported in this paper.

Also, some manufacturing tasks *require* a haptic feedback (training applications for example). We would like to study the benefits of continuous collision detection algorithms in this context.

References

[1] M. Anitescu and F. A. Potra. Formulating Dynamic Multi-Rigid-Body Contact Problems with Friction as Solvable Linear Complementarity Problems. Nonlinear Dynam. 14 (1997), no. 3, 231–247.

[2] M. Anitescu, F. A. Potra and D. E. Stewart. Time-stepping for Three-dimensional Rigid Body Dynamics. Computational Modeling of Contact and Friction. Comput. Methods Appl. Mech. Engrg. 177 (1999), no. 3-4, 183–197.

[3] D. Baraff. Fast Contact Force Computation for Nonpenetrating Rigid Bodies. In SIGGRAPH 94 Conference Proceedings, Annual Conference Series, pp 23-34. ACM SIGGRAPH, Addison Wesley, 1994.

[4] D. Baraff. Interactive Simulation of Solid Rigid Bodies. IEEE Computer Graphics and Applications, 15(3), pp 63-75, May 1995.

[5] J. Baumgarte. Stabilization of constraints and integrals of motion in dynamical systems. Comp. Meth. in Appl. Mech. and Eng., 1:1-16, 1972.

[6] G. Bradshaw and C. O'Sullivan. Sphere-Tree Construction using Dynamic Medial Axis Approximation. In Proceedings of ACM Symposium on Computer Animation 2002.

[7] J. F. Canny. collision detection for moving polyhedra. IEEE Trans. Patt. Anal. Mach. Intell. 8,2 (March 1986), pp 200-209.

[8] S. Gottschalk, M. C. Lin, and D. Manocha. OBB-Tree: A Hierarchical Structure for Rapid Interference Detection. In SIGGRAPH 96 Conference Proceedings, Annual Conference Series. ACM SIGGRAPH, Addison Wesley, August 1996.

[9] J.T. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral and K. Zikan. Efficient collision Detection Using Bounding Volume Hierarchies of k-DOPs. IEEE Transactions on Visualization and Computer Graphics, March 1998, Volume 4, Number 1.

[10] P. M. Hubbard. Collision detection for interactive graphics applications. Ph.D. Thesis, April 1995.

[11] P. Jiménez, F. Thomas and C. Torras. 3D collision detection: a survey. Computers and Graphics, 25 (2), pp. 269-285, (April 2001), Pergamon Press / Elsevier Science.

[12] M. McKenna and D. Zeltzer. Dynamic simulation of autonomous legged locomotion. In Computer Graphics (Proc. SIGGRAPH), volume 24, pages 29-38. ACM, August 1990.

[13] Ming Lin and Dinesh Manocha. Collision and Proximity Queries. Handbook of Discrete and Computational Geometry: Collision detection, 2003 (to appear).

[14] P. Lötstedt. Numerical simulation of time-dependent contact friction problems in rigid body mechanics. SIAM Journal of Scientific Statistical Computing, vol. 5, no. 2, pp. 370- 393, 1984.

[15] V. J. Milenkovic and H.Schmidl. Optimization Based Animation. SIGGRAPH 2001

[16] B. Mirtich and J. Canny. Impulse-based Simulation of Rigid Bodies. In Proceedings of Symposium on Interactive 3D Graphics, 1995.

[17] R. E. Moore. Interval analysis and automatic error analysis in digital computation. PhD Thesis, Stanford University, October 1962.

[18] M. Moore and J. Wilhelms. Collision Detection and Response for Computer Animation. In Computers Graphics (Proceedings of SIGGRAPH 88), Annual Conference Series, pp 289-298. ACM SIGGRAPH, August 1988.

[19] S. Quinlan. Efficient distance computation between non-convex objects. In Proceedings of International Conference on Robotics and Automation, pp 3324-3329, 1994.

[20] S. Redon, A. Kheddar, and S. Coquillart. An algebraic solution to the problem of collision detection for rigid polyhedral objects. Proc. of IEEE Conference on Robotics and Automation, 2000.

[21] S. Redon, A. Kheddar and S. Coquillart. CONTACT: arbitrary in-between motions for continuous collision detection. In Proceedings of IEEE ROMAN'2001, Sep. 2001.

[22] S. Redon, A. Kheddar and S. Coquillart. Gauss' least constraint principle and rigid body simulations. In Proceedings of IEEE International Conference on Robotics and Automation, May 2002.

[23] S. Redon, A. Kheddar, and S. Coquillart. Fast continuous collision detection between rigid bodies. Proc. of Eurographics (Computer Graphics Forum), 2002.

[24] D. Ruspini and O. Khatib. Collision/Contact Models for the Dynamic Simulation of Complex Environments. IEEE/RSJ International Conference on Intelligent Robots and Systems:IROS'97.

[25] D. C. Ruspini, K. Kolarov and O. Khatib. The Haptic Display of Complex Graphical Environments. Computer Graphics Proceedings, SIGGRAPH 97 pp 345-52

[26] J. Snyder. Interval analysis for Computer Graphics. Computer Graphics, 26(2), pages 121-130, July 1992.

[27] D. E. Stewart and J. C. Trinkle. An Implicit Time-Stepping Scheme for Rigid Body Dynamics with Inelastic collisions and Coulomb Friction. International Journal of Numerical Methods in Engineering, 39:2673-2691, 1996.

[28] G. Van den Bergen. Efficient collision detection of complex deformable models using AABB trees. Journal of Graphics Tools, 2(4):1-14, 1997.

[29] D. R. Wilhelmsen. A Nearest Point Algorithm for Convex Polyhedral Cones and Applications to Positive Linear Approximations. Mathematics of computation, 30, pp 48-57, 1976.